Reliability Guaranteed Energy-Aware Frame-Based Task Set Execution Strategy for Hard Real-Time Systems

Zheng Li[a], Li Wang[a], Shuhui Li[a], Shangping Ren[a], Gang Quan[b]

[a]*Illinois Institute of Technology, Chicago, IL 60616, USA*
[b]*Florida International University, Miami, FL 33174, USA*

Abstract

In this paper, we study the problem of how to execute a real-time frame-based task set on DVFS-enabled processors so that the system's reliability can be guaranteed and the energy consumption of executing the task set is minimized. To ensure the reliability requirement, processing resources are reserved to re-execute tasks when transient faults occur. However, different from commonly used approaches that the reserved processing resources are shared by all tasks in the task set, we judiciously select a subset of tasks to share these reserved resources for recovery purposes. In addition, we formally prove that for a give task set, the system achieves the highest reliability and consumes the least amount of energy when the task set is executed with a uniform frequency (or neighboring frequencies if the desired frequency is not available). Based on the theoretic conclusion, rather than heuristically searching for appropriate execution frequency for each individual task as used in many research work, we directly calculate the optimal execution frequency for the task set. Our simulation results have shown that with our approach, we can not only guarantee the same level of system reliability, but also have up to 15% energy saving improvement over other fault recovery-based approaches existed in the literature. Furthermore, as our approach does not require frequent frequency changes, it works particularly effective on processors where frequency switching overhead is large and not negligible.

Keywords: Transient fault, real-time, energy saving, frame-based task set

1. Introduction

Reliability and power/energy efficiency have increasingly become one of the critical issues in real-time system designs. As aggressive scaling of CMOS technology continues, transistors also become more vulnerable than ever before to radiation related external impacts [5, 7, 14], which often lead to rapid system reliability degradation. The system reliability problem is further worsened by ever increasing system complexity and dramatically increased operating temperature [8, 15]. As a result, computing systems become more and more susceptible to both transient and permanent faults [15].

Furthermore, as more and more transistors are integrated into a single chip, operation power consumption of the chip has also increased exponentially. The dramatic power consumption increase not only exacerbates the power supply problem for power constrained platforms such as portable devices, but also aggravates the operational cost for power-rich platforms such as data centers. Therefore, more and more aggressive power aware reduction techniques have been proposed [6, 9].

However, for real-time system, power/energy conservations and reliability enhancement are usually at odds. Taking the Dynamic Voltage and Frequency Scaling (DVFS) technique as an example, as we know the DVFS is one of the widely used means for power management [13], by dynamically lowering down the supply voltage and working frequency, the DVFS can reduce system's power consumption. However, reducing the system's working frequency may also prolong task execution times and potentially cause tasks to miss their deadlines. In addition, existing work [23] has shown that the transient fault rate increases when the supply voltage on an IC chip is scaled down. In other words, lowering down system's supply voltage can potentially degrade the system's reliability. Hence, for systems demanding reliability and deadline guarantees and low energy consumption, such as satellite and surveillance systems [20], it is a challenge to design a task execution strategy that guarantees the satisfaction of both system reliability and deadline requirements, but at same time consumes the least amount of energy.

As transient faults are found more frequent than permanent faults [4, 9], in this work, we focus on the transient fault and explore using backward fault recovery techniques. The backward fault recovery technique stores system's safe state. In case of a failure occurrence, the system is restored to its previous safe state and the computation after the safe state point is repeated [12], to tolerate transient faults and guarantee system reliability requirement. As both the backward fault recovery techniques for reliability guarantee and the DVFS for energy savings depend on task's available slack time, i.e., the temporal difference between a task's deadline and its completion time, which is limited for hard real-time system, therefore, the problem we are interested is, how to effectively utilize the limited slack time to design a system with guaranteed reliability and

Email addresses: `zli80@iit.edu` (Zheng Li), `lwang64@iit.edu` (Li Wang), `sli38@iit.edu` (Shuhui Li), `ren@iit.edu` (Shangping Ren), `gang.quan@fiu.edu` (Gang Quan)

minimum energy consumption.

A few researches have been published in the literature which are closely related to our work. For example, Zhu et al. [19] proposed a reliability-aware power management scheme which aims to minimize energy consumption while maintaining system's reliability at the same level as if all tasks were executed at the highest processor frequency. However, as scaling down task execution frequencies for energy saving purpose often degrades system's reliability, hence, the basic idea behind the scheme is to reserve some slack time exclusively for fault recovery to maintain system's reliability at the target level. We call the reserved slack time for fault recovery as *recovery block*. A few heuristics as to how to decide the size of *recovery block* and the task execution frequencies, such as the longest task first (LTF) and the slack usage efficiency factor (SUEF), are developed [20]. Zhao et al. [17] improved the approach and developed the *shared recovery* (SHR) technique which allows all tasks share the same reserved recovery block, but only allow a single fault recovery during the entire task set execution.

Recently, Zhao et al. [18] extended the work and developed the *Generalized Shared Recovery Technique* (GSHR) approach which allows multiple fault recoveries by reserving multiple recovery blocks. As the GSHR approach allows sharing the recovery block among different tasks, the slack time is more efficiently used and thus has a better energy saving performance comparing to its predecessors. Zhao et al also developed a heuristic, i.e. the *Incremental Reliability Configuration Search* (IRCS), to determine the processor frequencies for different tasks. The IRCS essentially adopts a dynamic programming approach and searches for a suitable frequency for each task to maximize energy savings.

The techniques mentioned above work well when task execution times in a given task set do not have large variations and the available slack time, i.e., the temporal difference between the deadline and its earliest completion time of the task set, is large. However, when the slack time is limited, if there is a task whose execution time is much larger than the other tasks in the task set, the aforementioned approaches need to reserve a recovery block long enough to accommodate the outlier task, which not only results in low usage of the reserved block for short tasks, but also less slack time can be used to scale down the processing frequency for energy saving purposes.

In this paper, we present the *Generalized Subset Shared Recovery* technique (GSSR) to guarantee hard real-time system's reliability and at the same time minimize energy consumption of executing a given task set. The essence of the developed GSSR approach is to judiciously partition a given task set into two subsets: *fault-unprotected task set* and *fault-protected task set*. To guarantee the reliability, the fault-unprotected task set is executed with the highest processor frequency; while the execution frequencies for fault-protected tasks are scaled down and tasks in the fault-protected task set share recovery block in case of a failure occurrence. As such, our approach can effectively reduce unnecessary resource reservations and leave more slack time to adjust tasks' processing frequencies for energy saving.

We also formally prove that, if a task set remains active in an interval, executing the task set under a uniform frequency is the optimal strategy to maximize task set reliability and minimize the energy consumption. If such a frequency is not available, then using two neighboring frequencies, i.e. adjacent frequencies, become the optimal choice. Therefore, different from the IRCS that different tasks may run at different frequencies, we employ two (if the desired scaled-down frequency is available), or at most three, different processor frequencies for the entire task set. Our extensive experimental study has shown that, comparing to recent work in the literature, such as the IRCS, the proposed GSSR approach can achieve up to 15% more energy saving without sacrificing reliability and deadline guarantees.

The remainder of the paper is organized as follows: Section 2 introduces the system models and definitions. In Section 3, we present a motivating example and formally formulate the task set execution strategy (TSES) decision problem to be addressed. We discuss the proposed approach in Section 5. Section 6 presents the performance evaluation. We conclude the paper in Section 7.

2. Models and Definitions

In this section, we introduce the models and terms used in the paper.

2.1. Processor and Task Model

We assume the processor is DVFS enabled and has a finite set (F) of working frequencies, i.e. $F = \{f_1, ..., f_q\}$ with $f_i < f_j$ if $i < j$, where f_i and f_{i+1} are called neighboring frequencies. The minimum and maximum frequency f_1 and f_q are denoted as $f_{\min}$ and $f_{\max}$, respectively. The values are normalized with respected to $f_{\max}$, i.e. $f_{\max} = 1$.

As the processor's processing frequency is almost linearly related to supply voltage [1] and the supply voltage adjustment will result in processing frequency scaling, hence, in this paper, we use frequency scaling to stand for both supply voltage and frequency adjustment.

The task set Γ being considered is a frame-based task set [2]. It has n tasks, i.e. $\Gamma = \{\tau_1, \cdots, \tau_i, \cdots, \tau_n\}$ with $c_i \geq c_j$ if $i < j$ where c_i is the worst-case execution time (WCET) of task τ_i under the maximum frequency $f_{\max}$. When a lower working frequency f_i ($f_i < f_{\max}$) is used, the execution time of a task is extended to $\frac{c_i}{f_i}$. For simplicity, we use $f(\tau_i)$ to denote the execution frequency of task τ_i.

Tasks in a given task set may or may not be independent. To ease the representation, we first assume that tasks are independent, then in Section 5.3, we explain why the proposed technique can also be applied to a dependent task set.

[1] $f = a \cdot \frac{(V_{dd}-V_t)^2}{V_{dd}}$, where a is constant, f, V_{dd} and V_t are the processing frequency, supply voltage and threshold voltage, respectively [2].

[2] A frame-based task set is a task set of which all tasks have the same arrival time and deadline [16].

2.2. *Energy Model*

The energy model used in this paper is adopted from Aydin et al.'s work [1, 21, 23]. For self containment, we give a short summary here.

In particular, the system's active power (P_a) under operating frequency f consists of frequency independent and frequency dependent power, i.e.

$$P_a = P_{\text{ind}} + P_{\text{dep}} = P_{\text{ind}} + C_{\text{ef}} f^{\theta} \quad (1)$$

where P_{ind}, C_{ef}, and θ are system dependent constants and $\theta \geq 2$ [1, 2]. If the task τ_i is executed under $f(\tau_i)$, its power consumption is represented as:

$$\begin{aligned} E(f(\tau_i), c_i) &= (P_{\text{ind}} + C_{\text{ef}} f(\tau_i)^{\theta}) \frac{c_i}{f(\tau_i)} \\ &= P_{\text{ind}} \frac{c_i}{f(\tau_i)} + C_{\text{ef}} c_i f(\tau_i)^{\theta - 1} \end{aligned} \quad (2)$$

From (2), it is clear that scaling down the processing frequency reduces frequency dependent energy ($C_{\text{ef}} c_i f(\tau_i)^{\theta-1}$) but it also increases the frequency independent energy ($P_{\text{ind}} \frac{c_i}{f(\tau_i)}$) because of longer execution time due to lower processing frequency. Hence, there is a balanced point, i.e. the *Energy-Efficient Frequency* (f_{ee}). Further scaling down the processing frequency below f_{ee} will increase the total energy consumption. Early studies [21, 23] concludes that

$$f_{ee} = \sqrt[\theta]{\frac{P_{\text{ind}}}{C_{\text{ef}}(\theta - 1)}} \quad (3)$$

2.3. *Fault and Reliability Model*

Although both permanent and transient faults may occur during the execution of task sets, transient faults are found more frequent than permanent faults [4, 9]. Hence, in this paper, we focus only on transient faults and adopt the same fault arrival rate model as given in [18, 20, 23]:

$$\lambda(f) = \hat{\lambda}_0 10^{-\hat{d} f} \quad (4)$$

where $\hat{\lambda}_0 = \lambda_0 10^{\frac{d}{1-f_{\text{min}}}}$, $\hat{d} = \frac{d}{1-f_{\text{min}}}$, d (> 0) is a system-dependent constant, and λ_0 is the average fault arrival rate under f_{max}.

The reliability of task τ_i under execution frequency $f(\tau_i)$ is defined as the probability of finishing its execution without encountering any transient faults, which is expressed as [18, 20]:

$$\gamma(f(\tau_i), c_i) = e^{-\lambda(f(\tau_i)) \frac{c_i}{f(\tau_i)}} \quad (5)$$

where $\lambda(f(\tau_i))$ is given by Equation (4).

2.4. *Definitions*

In this subsection, we introduce the terms and notations to be used later in the paper.

Reserved recovery block (B): a reserved recovery block is a time block reserved exclusively for task re-execution in the presences of faults.

If the reserved recover block B is for task τ_i, the length of the block is equal to τ_i's WCET under f_{max}, i.e. $len(B) = c_i$. If k (≥ 1) reserved recovery blocks are shared among tasks in the task set Γ, the total duration of the reserved blocks equals to the sum of the first k longest tasks' WCETs under f_{max}, i.e. $\sum_{i=1}^{k} len(B_i) = \sum_{i=1}^{k} c_i$, where $c_i \geq c_j$ if $i < j$.

Fault-Protected Task (τ^p): a fault-protected task is a task that is allowed to share a reserved recovery block for re-execution if a fault occurs during its execution.

Fault-Protected Task Set (Γ_p): a fault-protected task set is a task set of which all its member tasks are fault-protected, i.e. $\forall \tau \in \Gamma_p$, τ is fault-protected.

Fault-Unprotected Task (τ^u): a fault-unprotected task is the task which is *not* allowed to use a reserved recovery block for fault recovery even if faults occur during its execution.

Fault-Unprotected Task Set (Γ_u): a fault-unprotected task set is a task set of which all its member tasks are fault-unprotected. We have $\Gamma_u \cup \Gamma_p = \Gamma$ and $\Gamma_u \cap \Gamma_p = \emptyset$.

Task Set Execution Strategy (Ψ): for a given task set Γ, the task set execution strategy decides the number of reserved recovery blocks (k), the protect task set (Γ_p), and execution frequency for task τ_i. In other words, $\Psi(\Gamma) = (\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)$, where $\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}$ is the abbreviation for $[(f(\tau_1), \frac{c_1}{f(\tau_1)}), \cdots, (f(\tau_n), \frac{c_n}{f(\tau_n)})]$, i.e. executing task τ_i with frequency $f(\tau_i)$ for $\frac{c_i}{f(\tau_i)}$ time duration, $f(\tau_i) \in F$, and $\Gamma_p \subseteq \Gamma$.

Task Set Reliability under a Given Execution Strategy (R): for a given task set (Γ) and its processing strategy ($\Psi(\Gamma) = (\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)$), the task set reliability (R) is defined as the probability of successfully executing all tasks under the given execution strategy.

3. Motivation and Problem Formulation

Before we formulate the research problem, we first give a motivating example.

3.1. *A Motivating Example*

Considering a task set with four independent tasks, i.e. $\Gamma = \{A, B, C, D\}$, their execution times under f_{max} are $10ms$, $5ms$, $4ms$ and $3ms$, respectively. These tasks arrive at the same time and have the same deadline $35ms$. The reliability constraint for the task set is $R_g = \prod_{\tau_i \in \{A,B,C,D\}} \gamma(f_{\text{max}}, c_i)$, which is the probability of successfully executing all four tasks under f_{max} with zero fault occurrence. Fig. 1(a) depicts the setting.

Assume the available frequencies on a processor are from $f_{\text{min}} = 0.1$ to $f_{\text{max}} = 1$ with step value of 0.1, transient fault arrival follows Poisson distribution with average arrival rate as $\lambda_0 = 10^{-6}$ as given in [18]. We evaluate system reliability and energy consumption of two well recognized reliability-aware power management approaches, i.e. the approaches with [18] and without [20] reserved recovery block sharing, respectively.

Without Sharing Reserved Recovery Block — the Longest Task First (LTF) Approach [20]

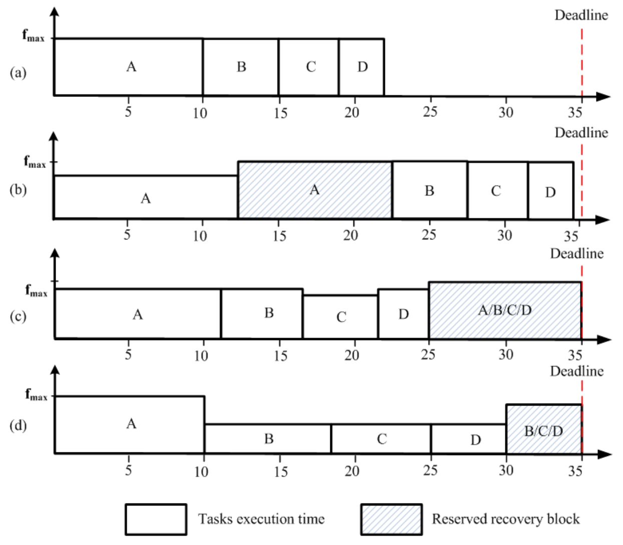

Figure 1: Motivating Example

The LTF approach, as the name suggested, always chooses the task with longest worst-case execution time in the task set and applies frequency scaling to the task. For the given example, as task A has the longest WCET of $10ms$, it is chosen for using frequency scaling. A reserved recovery block of size $10ms$ is allocated for task A in case faults occur during its execution. The other three tasks are executed with maximal frequency $f_{\max} = 1$. The remaining slack time of $3ms$ is hence used for scaling down the processing frequency for task A. In this case, task A should be executed under frequency of $10/13 = 0.7692$. Hence, the nearest available frequency 0.8 is chosen. Fig. 1(b) illustrates the execution strategy under LTF. It is not difficult to calculate that the total energy cost, normalized to the energy cost when all tasks are executed with $f_{\max}$, is 84%. The task set reliability, normalized to the reliability requirement of R_g, is 100.0007%.

Globally Sharing Reserved Recovery Block — the Generalized Shared Recovery (GSHR) Approach [18]

With the GSHR approach, reserved recovery blocks are shared among all tasks in the task set. The Incremental Reliability Configuration Search (IRCS) heuristic is developed [18] to search a suitable frequency for each task. In particular, task A, B, C and D are executed under frequency of 0.9, 0.9, 0.8 and 0.9, respectively. The recovery block is shared by all four tasks and has the length of $10ms$. Fig. 1(c) depicts the result of IRCS. The normalized energy consumption under the IRCS scheme is 79% and the normalized task set reliability is 100.0189%.

The example shows that the results derived from both the LTF and the IRCS satisfy the reliability and deadline requirements. However, the IRCS approach performs better than the LTF approach from energy consumption perspective.

Our Observations

In order to allow all tasks be able to share the same recovery block, the IRCS approach needs to reserve $10ms$ as its recovery block. If, on the other hand, we exclude the longest task, i.e. task A, from sharing the recovery block, we only need to reserve $5ms$ for the recovery block, which means more slack time can be used for frequency scaling. It is very interesting that, when we run task A at the maximum frequency and task B, C, and D at frequency 0.6, the normalized energy consumption is only 68% and the normalized reliability is 100.0110%. Under this execution strategy, we not only satisfy the reliability and deadline requirements, but also save 11% more energy comparing to the IRCS approach. Fig. 1(d) shows the processing strategy that outperforms the IRCS approach.

This example indicates that if we judiciously select a subset of tasks to share the recovery block, we can save more energy without compromising reliability and deadline constraints.

Another observation is that the IRCS uses a heuristic approach to searching a suitable frequency for each task in the task set. When the search space, i.e. the number of available frequencies and the number of tasks in the task set, is large, due to the heuristic nature, the chance of finding a good solution for every task reduces and hence the performance of the IRCS approach degrades. Simulation results in Section 6 confirm this observation. Therefore, our second motivation is to develop a task set execution strategy search algorithm which is independent of the number of available frequencies and the number of tasks in a task set.

3.2. Problem Formulation

In this subsection, we formulate the research problem the rest of the paper to address, i.e. to find a reliability and deadline guaranteed task set execution strategy that minimizes energy consumption. We call the problem the Task Set Execution Strategy decision problem, the TSES problem for short.

Problem 1 (The TSES Decision Problem)**.** *Given a task set with n independent tasks, i.e. $\Gamma = \{\tau_1, \cdots, \tau_n\}$ with $c_i \le c_j$ if $i > j$ where c_i is the τ_i's WCET under $f_{\max}$, and a DVFS enabled processor with q different processing frequencies, i.e. $F = \{f_1, \cdots, f_q\}$, where $f_1 = f_{\min}$, $f_q = f_{\max}$, and $f_i < f_j$ if $i < j$. The reliability and deadline constraints are R_g and D, respectively, decide a task set processing strategy $\Psi(\Gamma) = (\overrightarrow{f(\tau_i)}_{\tau_i\in\Gamma}, \Gamma_p, k)$ with*
Objective:

$$\min \sum_{\tau_i\in\Gamma} E(f(\tau_i), c_i) \tag{6}$$

Subject to:

$$R(\overrightarrow{f(\tau_i)}_{\tau_i\in\Gamma}, \Gamma_p, k) \ge R_g \tag{7}$$

$$\sum_{\tau_i\in\Gamma} \frac{c_i}{f(\tau_i)} \le D - \sum_{i=1}^{k} len(B_i) \tag{8}$$

where $len(B_i) = c_i^p$ is the ith longest WCET of fault-protected task $\tau_i^p \in \Gamma_p$, and $\Gamma_p \subseteq \Gamma$. □

It is worth pointing out that we assume fault detections are done at the end of each task execution using techniques such as sanity or consistency checks [12] and the time overhead is counted as part of task's worst-case execution time. Although, frequency switching has extra time and energy overhead, we leave the frequency switching overhead discussion to Section 5.3.

4. Generalized Subset Shared Recovery

As we have observed from the motivating example given in Section 3.1 that recovery block sharing has its superiority over the techniques that do not share the recovery block. When tasks' worst-case execution times in a given task set are close to each other, i.e. the variation among task's WCETs is small, globally sharing a reserved recovery block among all tasks in the given task set performs well as evidenced in [18]. However, when there is a large variation among the worst-case execution times, as shown in the motivating example, sharing the reserved recovery block among a judiciously selected subset of tasks performs better than globally sharing the block among all tasks in the task set. The intuition behind the conclusion is as follows.

For a given task set Γ of size n, when the variation of their worst-case execution times is large, k $(1 \leq k \leq n)$ reserved recovery blocks may occupy most of the slack time, hence leave only a small portion for frequency scaling. However, if we exclude the outliers, for instance, the first few longest tasks, and only allow a subset of tasks to share the recovery blocks, we can reduce the total length of the recovery blocks and leave more slack time for energy saving. Based on the intuition, we propose a *generalized subset shared recovery* (GSSR) technique. It is not difficult to see that the GSHR is a special case of the proposed GSSR technique, i.e. when $\Gamma_p = \Gamma$, the GSSR degenerates to the GSHR.

With the GSSR technique, a given task set Γ is partitioned into two subsets, i.e. fault-protected task set Γ_p and fault-unprotected task set Γ_u, where all fault-unprotected tasks are executed under the maximum frequency $f_{\max}$, i.e. $\forall \tau_u \in \Gamma_u$, $f(\tau_u) = f_{\max}$. Furthermore, fault-unprotected tasks are not allowed to use the reserved recovery blocks to do fault recoveries. On the other hand, tasks in the fault-protected task set are executed under scaled down frequencies and share the reserved recovery blocks for fault recoveries.

Given a task set Γ, and an execution strategy $\Psi(\Gamma) = (\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)$, where $\Gamma_p = \{\tau_1^p, \cdots, \tau_m^p\}$, reliability of the task set Γ is the product of the reliability of fault-protected task set and the reliability of fault-unprotected task set, i.e.

$$R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k) = R^k(\Gamma_p) \times \prod_{\tau_j \in \Gamma - \Gamma_p} \gamma(f(\tau_j), c_j) \quad (9)$$

where the calculation of the reliability of fault-protected task set sharing k reserved recovery blocks ($R^k(\Gamma_p)$) is recursively defined given below [18]:

$$R^k(\tau_1^p, \cdots, \tau_m^p) = \gamma(f(\tau_1^p), c_1^p) \times R^k(\tau_2^p, \cdots, \tau_m^p) + \\ (1 - \gamma(f(\tau_1^p), c_i^p)) \times \gamma(f_{\max}, c_1^p) \times R^{k-1}(\tau_2^p, \cdots, \tau_m^p) \quad (10)$$

The value of $R^k(\tau_1^p, \cdots, \tau_m^p)$ can be found using dynamic programming and the time complexity is $O(km)$.

As the probability of fault occurrences is relatively small, the expected energy consumption for fault recoveries is negligible comparing to the energy cost of executing all tasks in the task set. Hence, the expected energy consumption under given $\Psi(\Gamma) = (\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)$ can be defined as:

$$\mathcal{E}((\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)) = \sum_{\tau_i \in \Gamma} E(f(\tau_i), c_i) \quad (11)$$

From formula (9) and (11), we can see that the task set execution strategy Ψ has direct impact on the task set reliability and energy consumption. Next section, we present our approach to finding a task execution strategy that minimizes the energy consumption and at the same time guarantees the satisfaction of both reliability and deadline constraints.

5. GSSR-based Task Set Execution Strategy

As we have observed from our motivating example that if we judiciously select a fault-protected task set ($\Gamma_p \subseteq \Gamma$) and corresponding task execution frequencies for Γ_p, we can save more energy without compromising the satisfaction of reliability and deadline constraints. In this section, we present in detail the generalized subset shared recovery (GSSR) based task set execution strategy. In particular, we first consider for a given fault-protected task set (Γ_p), what should be its execution strategy $(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)$. Second, for a given task set (Γ), how to judiciously decide the fault-protected task set ($\Gamma_p \subseteq \Gamma$).

5.1. *Deciding Task Set Execution Strategy for a Given Fault-Protected Task Set*

In this subsection, we discuss for a given task set (Γ), if a judiciously selected fault-protected task set ($\Gamma_p \subseteq \Gamma$) is given, what processing frequencies $\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}$ should be used to execute the fault-protected tasks and how many blocks (k) need be reserved for shared recovery among all fault-protected tasks ($\tau_p \in \Gamma_p$).

As fault-unprotected tasks are not allowed to do fault recoveries, to achieve their highest reliability, they are always executed under $f_{\max}$. Hence, we only need to determine the execution frequencies for fault-protected tasks, i.e. $\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma_p}$. Before we consider the execution frequencies for a given task set, we first investigate the execution strategy for a single task without considering fault recovery.

The following lemma states that among all execution strategies taking the same amount of time T to complete a task's execution, the execution strategy that uses a uniform frequency, or neighboring frequencies if the desired frequency is not available, to execute the task for the fixed duration T provides the maximal reliability (γ), i.e. the maximal probability of successfully executing a task without encountering a fault.

Lemma 1. *Given a task τ, its WCET c, time duration T ($T \geq c$), and available frequencies $F = \{f_{\min}, f_2 \cdots, f_{\max}\}$ with $f_i < f_j$ for $i < j$, if frequency switching is allowed during task τ's execution, we have the following conclusions:*

1. *if* $f_u = f_{\max}\frac{c}{T} \in F$*, then*

$$\gamma(f_u, T) = \max\{\prod_{i=1}^{q} \gamma(f_i, t_i) | f_i \in F, \sum_{i=1}^{q} t_i = T\}$$

2. *otherwise*

$$\gamma([(f_j, t), (f_{j+1}, T-t)])$$
$$= \max\{\prod_{i=1}^{q} \gamma(f_i, t_i) | f_i \in F, \sum_{i=1}^{q} t_i = T\}$$

where $f_j < f_u < f_{j+1}$, $f_j, f_{j+1} \in F$ *and* $\sum_{i=1}^{q} f_i t_i = f_{\max} c$.

□

Proof. We use notation $\overrightarrow{(f_i, t_i)}_q = [(f_1, t_1), ..., (f_q, t_q)]$ to denote the strategy of executing a task with frequency $f_i \in F$ for t_i time units, where $t_i \geq 0$, $\sum_{i=1}^{q} f_i t_i = f_{\max} c$ and $\sum_{i=1}^{q} t_i = T$. Hence, the reliability of the task under $\overrightarrow{(f_i, t_i)}_q$ can be represented as:

$$\gamma(\overrightarrow{(f_i, t_i)}_q) = \prod_{i=1}^{q} \gamma(f_i, t_i) = e^{-\sum_{i=1}^{q} \lambda(f_i) t_i}$$

If $f_u = f_{\max}\frac{c}{T} \in F$ is used for the whole duration T, then the execution strategy is $(f_{\max}\frac{c}{T}, T) = (\frac{\sum_{i=1}^{q} f_i t_i}{\sum_{i=1}^{q} t_i}, \sum_{i=1}^{q} t_i)$, and the reliability is:

$$\gamma(f_{\max}\frac{c}{T}, T) = e^{-\lambda(\frac{\sum_{i=1}^{q} f_i t_i}{\sum_{i=1}^{q} t_i}) \sum_{i=1}^{q} t_i}$$

As $\lambda(x)$ is a convex function[3], based on convex function property (12), we have:

$$\sum_{i=1}^{q} \lambda(f_i) t_i \geq \lambda(\frac{\sum_{i=1}^{q} f_i t_i}{\sum_{i=1}^{q} t_i})(\sum_{i=1}^{q} t_i)$$

Furthermore, as e^{-x} is a decreasing function, we have $\gamma(f_{\max}\frac{c}{T}, T) \geq \gamma(\overrightarrow{(f_i, t_i)}_q)$, which concludes the proof for case 1).

If $f_{\max}\frac{c}{T} \notin F$ and $f_j < f_{\max}\frac{c}{T} < f_{j+1}$, the reliability of task under $[(f_j, t), (f_{j+1}, T-t)]$ can be expressed as:

$$\gamma([(f_j, t), (f_{j+1}, T-t)]) = e^{-(\lambda(f_j)t + \lambda(f_{j+1})(T-t))}$$

According to formula (12), we have:

$$\sum_{i=1}^{q} \lambda(f_i) t_i \geq \lambda(f_j) t + \lambda(f_{j+1})(T-t)$$

[3] If $g(x)$ is a convex function, $q(\geq 2) \in I^+$, $x_i, t_i \in \Re^+$ and $x_i < x_k$ if $i < k$, then we have:

$$\sum_{i=1}^{q} g(x_i) t_i \geq g(x_j) t_j + g(x_{j+1}) t_{j+1} \geq g(\frac{\sum_{i=1}^{q} x_i t_i}{\sum_{i=1}^{q} t_i})(\sum_{i=1}^{q} t_i) \quad (12)$$

Where $x_j t_j + x_{j+1} t_{j+1} = \sum_{i=1}^{q} x_i t_i, x_j < \frac{\sum_{i=1}^{q} x_i t_i}{\sum_{i=1}^{r} t_i} < x_{j+1}$, and $t_j + t_{j+1} = \sum_{i=1}^{q} t_i$.

Hence, $e^{-(\lambda(f_j)t + \lambda(f_{j+1})(T-t))} \geq e^{-\sum_{i=1}^{q} \lambda(f_i) t_i}$, which implies $\gamma([(f_j, t), (f_{j+1}, T-t)]) \geq \gamma(\overrightarrow{(f_i, t_i)}_q)$. These conclude the proof. □

Lemma 1 can easily be extended to a task set (Γ) by treating the single task's execution time c as the summation of c_i, i.e. $c = \sum_{\tau_i \in \Gamma} c_i$. We formulate the extension as Lemma 2.

Lemma 2. *Given a task set* $\Gamma = \{\tau_1, \cdots, \tau_n\}$ *with task* τ_i*'s worst-case execution time* c_i*, available frequency* $F = \{f_{\min}, f_2, \cdots, f_{\max}\}$ *with* $f_i < f_j$ *for* $i < j$*, and fixed execution time duration* T *(*$T \geq \sum_{\tau_i \in \Gamma} c_i$*), when shared recovery block* $k = 0$*, the execution strategy of using a uniform frequency* $f_u = \frac{f_{\max} \sum_{\tau_i \in \Gamma} c_i}{T}$ *if* $f_u \in F$*, or neighboring frequencies* f_i *and* f_{i+1} *with* $f_i < f_u < f_{i+1}$ *if* $f_u \notin F$*, to execute the task set provides the highest reliability. More specificly, we have*

1. *if* $f_u = f_{\max}\frac{\sum_{\tau_i \in \Gamma} c_i}{T} \in F$*, then*

$$R(\overrightarrow{(f_u, \frac{c_i}{f_u})}_{\tau_i \in \Gamma}, \phi, 0)$$
$$= \max\{R(\overrightarrow{(f(\tau_i), \frac{c_i}{f(\tau_i)})}_{\tau_i \in \Gamma}, \phi, 0) | f(\tau_i) \in F, \sum_{\tau_i \in \Gamma} \frac{c_i}{f(\tau_i)} = T\} \quad (13)$$

2. *otherwise,*

$$R([(f_j, t), (f_{j+1}, T-t)], \phi, 0)$$
$$= \max\{R(\overrightarrow{(f(\tau_i), \frac{c_i}{f(\tau_i)})}_{\tau_i \in \Gamma}, \phi, 0) | f(\tau_i) \in F, \sum_{\tau_i \in \Gamma} \frac{c_i}{f(\tau_i)} = T\} \quad (14)$$

where $f_j \times t + f_{j+1}(T-t) = \sum_{i=1}^{n} f_{\max} c_i$*, or* $t = \frac{\sum_{j=1}^{n} c_i - f_{j+1} \times T}{f_j - f_{j+1}}$*,* $f_j < f_u < f_{j+1}$ *and* $f_j, f_{j+1} \in F$

□

We skip the proof for Lemma 2 as it is very similar to the one given for Lemma 1.

Rizvandi [13] studied the relationship between task execution frequency and energy consumption and concluded that when the energy model is a convex function of frequency, using a uniform, or neighboring frequencies if the desired frequency is not available, is the optimal strategy for energy saving purpose. With this conclusion, we derive the Lemma 3.

Lemma 3. *Given a task set* $\Gamma = \{\tau_1, \cdots, \tau_n\}$ *with task* τ_i*'s worst-case execution time* c_i*, available frequency* $F = \{f_{\min}, f_2, \cdots, f_{\max}\}$ *with* $f_i < f_j$ *for* $i < j$*, and fixed execution time duration* T *(*$T \geq \sum_{\tau_i \in \Gamma} c_i$*), when shared recovery block* $k = 0$*, then using one uniform frequency* $f_u = \frac{f_{\max} \sum_{\tau_i \in \Gamma} c_i}{T}$*, or neighboring frequencies* f_j *and* f_{j+1} *with* $f_j < f_u < f_{j+1}$ *if* $f_u \notin F$*, to execute the task set consumes the least energy.* □

Proof. Since the execution time duration is fixed as T and the energy model (formula (2)) is a convex function of processing frequency, hence, executing the task set under frequency $f_u = \frac{f_{\max} \sum_{\tau_i \in \Gamma} c_i}{T}$, or neighboring frequencies f_j and f_{j+1} with $f_j < f_u < f_{j+1}$ if $f_u \notin F$, consumes the least energy [13]. In addition, based on Lemma 2, executing the task set under frequency f_u, or the neighboring frequencies f_j and f_{j+1} with $f_j < f_u < f_{j+1}$ if $f_u \notin F$, can also achieve the highest reliability. Hence, we get the conclusion summarized as Lemma 3. $\square$

From Lemma 2 and Lemma 3, we have the following theorem.

Theorem 1. *Given a task set $\Gamma = \{\tau_1, \cdots, \tau_n\}$ with task τ_i's worst-case execution time c_i, available frequency $F = \{f_{\min}, f_2, \cdots, f_{\max}\}$ with $f_i < f_j$ for $i < j$, and fixed execution time duration T ($T \geq \sum_{\tau_i \in \Gamma} c_i$), when shared recovery block $k = 0$, then the execution strategy of using a uniform frequency $f_u = \frac{f_{\max} \sum_{\tau_i \in \Gamma} c_i}{T}$ if $f_u \in F$, or neighboring frequencies f_j and f_{j+1} with $f_j < f_u < f_{j+1}$ if $f_u \notin F$, to execute the task set provides the highest reliability and consumes the least energy.* $\square$

Proof. The proof of the theorem can be directly derived from the proof of Lemma 2 and Lemma 3. $\square$

Theorem 1 gives the execution strategy that has the highest reliability, but consumes the least energy for a task set when zero recovery is considered. We argue that when k recovery blocks are allocated for fault-protected task set, the conclusions stated in Theorem 1 still holds.

The intuition behind it is that if a strategy under which executing the task set has the highest reliability, i.e. the probability to successfully execute the task set without encountering a fault, it simply implies that executing the task set with the strategy has the least probability to encounter a fault, hence, has the least probability to require a recovery. Therefore, when a given number of recovery blocks become available, the strategy will perform the best, i.e. provide the highest reliability among all other strategies. As task set execution energy consumption is independent of recovery blocks as shown in (11), the execution strategy that leads to the minimal energy consumption under zero recovery blocks will remain to be the optimal one when there are k reserved recovery blocks.

It is worth pointing out that when $f_u = \frac{f_{\max} \sum_{\tau_i \in \Gamma} c_i}{T} \notin F$, based on (14), we need to use f_j and f_{j+1} to execute for t and $T - t$ time units, respectively, where $f_j < f_u < f_{j+1}$, $f_j, f_{j+1} \in F$, and $t = \frac{\sum_{i=1}^{n} c_i - f_{j+1} \times T}{f_j - f_{j+1}}$. The question is if frequency switching is only allowed at the beginning of each task, and t happens to be in the middle of a task τ_m's execution time, i.e. $\sum_{i=1}^{m-1} c_i < f_j t < \sum_{i=1}^{m} c_i$, where $1 \leq m \leq n$. In this case, we choose a conservative approach, i.e. use higher frequency f_{j+1} to execute task τ_m to guarantee reliability.

Now, we are ready to give the task set execution strategy for a given fault-protected task set Γ_p with deadline D and reliability constraint R_g. The basic idea is we start with 0 recovery blocks, i.e. $k = 0$, and use Lemma 2 and (5) to calculate the reliability R with $T = D$. If $R \geq R_g$, then the execution strategy is $\Psi = (\overrightarrow{(f_u, \frac{c_i}{f_u})}_{\tau_i \in \Gamma_p}, \Gamma_p, 0)$, if $f_u = f_{\max} \frac{\sum_{\tau_i \in \Gamma} c_i}{T} \in F$; or $\Psi = (\overrightarrow{[(f_j, \frac{c_i}{f_j})_{i<m}, (f_{j+1}, \frac{c_i}{f_{j+1}})_{m \leq i \leq n}]}, \Gamma_p, 0)$. Otherwise, we allocate one recovery block, i.e. $k = 1$, then the available time for task execution is $T = D - c_1^p$, where $c_1^p = \max_{\tau_i \in \Gamma_p}\{c_i\}$ is the longest WCET in the fault-protected task set Γ_p. We then use Lemma 2 and (10) to calculate reliability R, if $R \geq R_g$, we have obtained the execution strategy with $k = 1$. Otherwise, we iterate the process with $k = i$ and $T = D - \sum_{j=1}^{i} c_j^p$ until the reliability is satisfied, where c_j^p is jth longest WCET in the fault-protected task set Γ_p.

We use an example to explain the procedure.

Example 1. *Assume a fault-protected task set has three tasks A, B, and C with WCET c_i under $f_{\max}$ of $8ms$, $6ms$, and $4ms$, respectively. The deadline for the task set is $D = 30ms$ and the reliability constraint is $R_g = \prod_{\tau_i \in \{A,B,C\}} \gamma(f_{\max}, c_i)$. The available discrete frequencies $F = \{0.1, 0.2, ..., 1\}$.*

Table 1: Execution Strategy for Fault-Protected Task Set

k	T	f_u	$(\tilde{f}_l, \tilde{f}_u)$	$f(\tau_i) = \tilde{f}_l$	$f(\tau_i) = \tilde{f}_u$	$\frac{R}{R_g}(\%)$
0	30	0.6	$(0.6, 0.6)$	A, B, C	A, B, C	99.850
1	22	0.8182	$(0.8, 0.9)$	A, B	C	100.001

The execution strategy is determined in two iterations. First, when no recovery block is reserved, i.e. $k = 0$, all tasks are assigned to frequency $f_u = \frac{8+6+4}{30} = 0.6$. Based on formula (9), the normalized reliability is 99.85%, which implies that the reliability constraint is not satisfied. Hence, we increase k to 1 and the desired frequency is $f_u = \frac{8+6+4}{30-8} = 0.8182 \notin F$. The neighboring frequencies $(0.8, 0.9)$ are used for executing the tasks. The calculated optimal frequency switching point is $t = 18$, which is found in the middle of task C's execution. Therefore, we execute A and B under 0.8 and C under 0.9. The normalized reliability under this execution strategy is 100.001% which satisfies the reliability requirement. Hence, the execution strategy is to reserve one recovery block of size 8ms, and execute task A and B with frequency 0.8, task C with frequency 0.9. Table 1 gives the execution detail.

The algorithm of using a uniform or neighboring frequencies scaling (UNS) for task set execution under a given fault-protected task set is illustrated in Algorithm 1.

We start from reserving zero recovery block, i.e, $k = 0$ (line 3) to find the uniform frequency f_u (line 4), if f_u is not available, using neighboring frequencies $\tilde{f}_u$ and $\tilde{f}_l$ instead (line 8 - 16). If the reliability requirement is satisfied, we stop and return the solution (line 17 - 19). Otherwise, reserving one more recovery block and repeat the above steps until reliability constraint is satisfied or slack time is used up (Line 3 - 21).

The time complexity of this algorithm is dominated by the `while` loop (line 3 - 21). The time complexity of calculating the reliability $R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)$ is $O(n^2)$ and the number of iterations of `while` loop (Line 3) is $O(n)$ since the number of tasks in the task set is n. Hence, the total time complexity should be $O(n^3)$.

5.2. *Deciding the Fault-Tolerated Task Set*

The previous subsection assumes the fault-tolerated task set (Γ_p) is given. In this subsection, we discuss how to judiciously select the set.

Algorithm 1 UNS(Γ_p, R_g, D, $F = \{f_1, ..., f_q\}$)

```
1:  $\tilde{f}_l = \tilde{f}_u = f_q, f(\tau_i) = f_q$
2:  $k = 0, T = D, m = |\Gamma_p|, slack = T - \sum_{i=1}^{m} c_i^p$
3:  while $slack > 0$ do
4:    $f_u = \frac{\sum_{i=1}^{m} c_i^p}{T}$
5:    $id = \min\{i | f_i \geq f_u \wedge f_i \in F\}$
6:    $\tilde{f}_u = f_{id}$
7:    $f(\tau_i) = \tilde{f}_u, \forall \tau_i^p \in \Gamma_p$
8:    if $\tilde{f}_u \neq f_u$ then
9:      $f_{ee} = \min\{f_i | f_i \geq f_{ee} \wedge f_i \in F\}$
10:     if $\tilde{f}_u > f_1 \wedge \tilde{f}_u > f_{ee}$ then
11:       $\tilde{f}_l = f_{id-1}$
12:       $t = \frac{\sum_{i=1}^{m} c_i^p - \tilde{f}_u T}{\tilde{f}_l - \tilde{f}_u}$
13:       $id = \max\{j | \sum_{i=1}^{j} c_i^p \leq \tilde{f}_l t\}$
14:       $f(\tau_i) = \tilde{f}_l, \forall i \leq id$
15:     end if
16:   end if
17:   if $R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma_p}, \Gamma_p, k) \geq R_g$ then
18:     break
19:   end if
20:   $k = k + 1, T = T - c_k^p, slack = slack - c_k^p$
21: end while
22: return $(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma_p}, k)$
```

Lemma 4. *For the given task set (Γ) and the reserved recovery blocks, we have:*

$$R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k) > R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma'_p, k) \text{ if } \Gamma'_p \subset \Gamma_p$$

where k is the number of reserved recovery blocks. □

Proof. Let $\Gamma_u = \Gamma - \Gamma_p$ and $\Gamma_v = \Gamma_p - \Gamma'_p = \{\tau_i, \tau_{i+1}, ..., \tau_j\}$ ($j > i$), then based on formula (10) , we have:

$$\begin{aligned} R^k(\Gamma_p) &> \gamma(f(\tau_i), c_i) \times R^k(\Gamma_p - \{\tau_i\}) \\ &> \prod_{l=i}^{i+1} \gamma(f(\tau_l), c_l) \times R^k(\Gamma_p - \{\tau_i, \tau_{i+1}\}) \\ &> ... \\ &> \prod_{\tau_l \in \Gamma_v} \gamma(f(\tau_l), c_l) \times R^k(\Gamma'_p) \end{aligned}$$

According to formula (9) , we have:

$$R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k) = R^k(\Gamma_p) \times \prod_{\tau_j \in \Gamma - \Gamma_p} \gamma(f(\tau_j), c_j)$$

and,

$$\begin{aligned} R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma'_p, k) &= R^k(\Gamma'_p) \times \prod_{\tau_l \in \Gamma_v} \gamma(f(\tau_l), c_l) \\ &\times \prod_{\tau_j \in \Gamma - \Gamma_p} \gamma(f(\tau_j), c_j) \end{aligned}$$

Hence, $R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k) > R(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma'_p, k)$. □

Based on Lemma 4, we should maximize the number of tasks that can share the reserved recovery blocks. If k recovery blocks are reserved, the total duration of the recovery blocks is the summation of WCETs of the first k longest tasks that share the recovery blocks. Hence, when we exclude the long tasks in the task set from sharing the recovery blocks to reduce the reserved recovery block size, at the same time, we need to include as many short tasks as possible to share the recovery blocks for maximizing the reliability without increasing the total recovery block size.

However, we do not have any prior knowledge as to how many long tasks should be excluded, hence, a heuristic approach is needed to find the answer. We first let the initial fault-protected task set Γ_p to be the given task set, i.e. $\Gamma_p = \Gamma$. At each iteration, we remove the longest task from Γ_p and let all the remaining tasks in Γ_p to share the recovery blocks. Hence, n different selections of Γ_p will be generated. For each Γ_p, we use Algorithm 1 (UNS) to determine its execution strategy and select the one that consumes the least energy. The details of this GSSR-UNS based iterative search (IS) algorithm is given in Algorithm 2.

Algorithm 2 GSSR-UNS-IS (Γ, R_g, D, F)

```
1:  $\Gamma_p = \Gamma$
2:  $f(\tau_i) = f_{\max}, \forall \tau_i \in \Gamma$
3:  $E_0 = +\infty$
4:  $\Psi(\Gamma) = (\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, 0)$
5:  for $j = 1$ to $|\Gamma|$ do
6:    $D' = D - \sum_{\tau_i \in \Gamma - \Gamma_p} c_i$
7:    $R'_g = \frac{R_g}{\prod_{\tau_i \in \Gamma - \Gamma_p} \gamma(f_{\max}, c_i)}$
8:    $(\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma_p}, k)$ = UNS($\Gamma_p$, $R'_g$, $D'$, $F$)
9:    if $\mathcal{E}((\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)) < E_0$ then
10:     $\Psi(\Gamma) = (\overrightarrow{f(\tau_i)}_{\tau_i \in \Gamma}, \Gamma_p, k)$
11:     $E_0 = \mathcal{E}(\Psi(\Gamma))$
12:   end if
13:   remove the longest task from $\Gamma_p$
14: end for
15: return $\Psi(\Gamma)$
```

Line 1 - 4 initialize the variables, and line 5 - 14 iteratively search for the best fault-protected task set Γ. When Γ_p is determined, the execution strategy $\Psi(\Gamma)$ is also obtained.

The time complexity of the algorithm is dominated by the `for` loop (line 5 - 14). The time cost for step 8 is $O(n^3)$. Hence, the overall time complexity is $O(n^4)$.

It is worth pointing out that though the idea of iterative search for the fault-protected task set is simple, simulation results given in Section 6 show the obtained results are near-optimal.

5.3. Discussion

Dealing with Dependent Tasks

When tasks have precedence constraints, for instance, the tasks derived from Strassen matrix multiplication and Fast

Fourier Transform (FFT) [3] applications, their execution order must satisfy the precedence constraints. As in our proposed GSSR-UNS-IS approach, the procedure of determining tasks' working frequencies is independent of task execution orders, hence, the GSSR-UNS-IS approach can be directly applied to tasks with dependencies.

Frequency Switching Overhead

It is well-known that DVFS is an effective way to reduce energy consumption. However, recent studies [10, 11] have noticed that frequency switching is not free. It has both time and energy consumption overhead. Furthermore, frequent frequency switching may decrease hardware components' life span [10]. As with the IRCS or the LTF approach, different tasks may need different execution frequencies, the frequency switching overhead is high. With our strategy, however, two, or at most three different frequencies are used to execute the whole task set, i.e. f_{max} for fault-unprotected tasks and one or two frequencies (if desired frequency is not available) for fault-protected tasks. If the tasks are independent, we can adjust the execution order such that tasks under the same frequency are executed successively, then at most two frequency changes are needed resulting in low frequency switching overhead which is another advantage of the proposed GSSR-UNS-IS approach.

If the task set has precedence constraints, in order to minimize frequency switching overhead, we need to re-consider the partition of fault-protected and fault-unprotected task sets, and the tasks' frequency assignment, which will be our future work.

6. Performance Evaluation

In this section, we intend to evaluate our proposed GSSR-UNS-IS approach by comparing it with the two baseline approaches, i.e., LTF [20] and IRCS [18], which are the representatives of no recovery block sharing, and globally shared recovery block (GSHR) based approaches, respectively.

6.1. Simulation Setting

In our simulations, the system parameters, such as, transient fault average arrival rate and power model, are listed in Table 2. These values are considered realistic and widely used by the research community [18, 22].

Table 2: Parameter Settings

λ_0	P_{ind}	d	C_{ef}	θ
10^{-6}	0.05	3	1	3

For each task set, its task worst-case execution times are randomly generated with values uniformly distributed in a predefined range. The energy consumption of executing a task set under different execution strategies is normalized to the energy cost when all tasks are executed with f_{max}. We set R_0 as the reliability of the task set when all the tasks are executed under the maximum frequency, i.e. $R_0 = \Pi_{\tau_i \in \Gamma}\gamma(f_{\text{max}}, c_i)$. We first set the reliability requirement as the one when all tasks are executed under f_{max}, i.e. $R_g = R_0$. We then vary the reliability requirement R_g to evaluate the sensitivity of our proposed approach to R_g change. The results shown in the following figures are the average values of repeating the experiments with $1,000$ different task sets.

We evaluate the performance of the approaches from the following five aspects:

1. processor utilization (U) impact on the approaches
2. the approaches' sensitivity to task worst-case execution time variation ($\mathcal{V}$)
3. the approaches' scalability with respect to the size of task set ($n = |\Gamma|$)
4. the approaches' scalability with respect to the size of task ($c_{\text{min}} = \min_{\tau_i \in \Gamma}\{c_i\}$)
5. the approaches' sensitivity with respect to the reliability requirement (R_g) change

where processor utilization is defines as

$$U = \frac{\sum_{\tau \in \Gamma} c_i}{D} \tag{15}$$

and the variation of task worst-case execution times is

$$\mathcal{V} = \sqrt{\frac{c_{max}}{c_{min}}} \tag{16}$$

where c_{min} and c_{max} are the minimum and maximum worst-case execution times of the given task set Γ, respectively, i.e. $c_{\text{max}} = \max_{\tau_i \in \Gamma}\{c_i\}$, $c_{\text{min}} = \min_{\tau_i \in \Gamma}\{c_i\}$.

As our proposed GSSR-UNS-IS approach consists of two algorithms, i.e. uniform/neighbroing frequencies scaling for task set execution (UNS) and iterative search for the fault-protected task set (IS), in order to provide an insight of performance of these two algorithms, before giving the above comparisons, we compare GSSR-UNS-IS with the following three approaches to inspect the UNS and IS algorithms.

- GSHR-UNS: use the proposed UNS approach to find task set execution frequencies assuming all tasks are fault-protected.
- GSHR-BF: use brute-force approach to find the optimal task execution frequencies assuming all tasks are fault-protected.
- GSSR-UNS-BF: use the proposed UNS approach for fault-protected task set frequency assignment and use brute-force approach to explore all the possible fault-protected task sets to find the optimal one.

The GSHR-UNS and the GSHR-BF have the same fault-protected task set but with different task set execution strategies. By comparing these two, we can evaluate the effectiveness of the UNS algorithm. The GSSR-UNS-IS and the GSSR-UNS-BF approaches have the same task set execution strategy, but with different fault-protected task set selection algorithms, by comparing them, we can assess our proposed fault-protected task set selection algorithm.

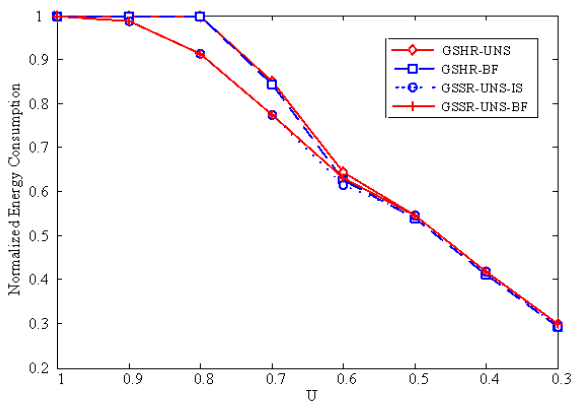

Figure 2: Evaluation of UNS and IS ($n = 5, c_{\min} = 20ms, \mathcal{V} = 4, R_g = R_0$)

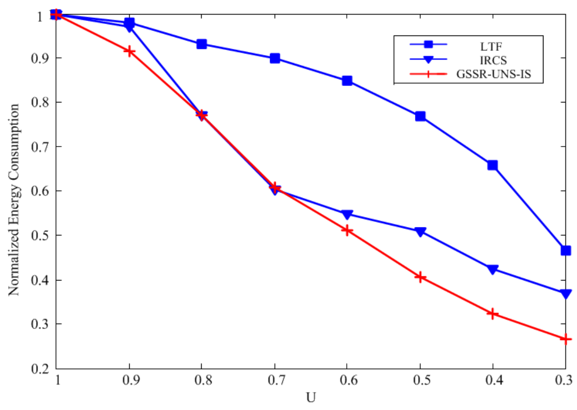

Figure 3: Processor Utilization Impact ($n = 20, c_{\min} = 20ms, \mathcal{V} = 4, R_g = R_0$)

6.2. Simulation Results and Discussions

6.2.1. Evaluation of UNS and IS

In this set of experiments, each task set has 5 tasks ($n = 5$), the minimum tasks' WCET is set as 20ms ($c_{\min} = 20$ms), the variation of task worst-case execution times is 4 ($\mathcal{V}$ = 4), the reliability requirement $R_g = R_0$. As brute-force approaches are timing-consuming, we can only limit the experiments with small task sets.

From the results depicted in Fig. 2, we can see the GSHR-UNS and the GSHR-BF nearly overlap with each other and the GSSR-UNS-IS and the GSSR-NS-BF are also overlapped, which indicates that our task set execution algorithm, i.e. UNS, is closed to the brute-force approach, and our fault-protected task set selection algorithm, i.e. IS, is near-optimal. In addition, we find that when $U > 0.6$, the GSSR based approaches, i.e. the GSSR-UNS-IS and the GSSR-UNS-BF approach have better energy saving performance than both the GSHR-UNS and the GSHR-BF approaches. This is due to the advantage of excluding some long tasks from fault protection, and hence increases the opportunities to scale down tasks' working frequencies for energy saving. However, when the slack time is long enough, fault-unprotected tasks will adversely restrict the energy saving performance as their working frequencies are not allowed to be adjusted. Under this case, the optimal fault-protected task set is the whole task set containing all tasks in the task set and the GSSR degenerates to the GSHR, hence, these four algorithms have the similar energy saving performance when $U < 0.6$.

6.2.2. Comparison of GSSR-UNS-IS *with LTF and IRCS*

In the following set of experiments, each task set has 20 tasks, the minimum tasks' WCET is set as 20ms, the variation of task worst-case execution times is 4 and the reliability requirement is set as $R_g = R_0$.

When a specific aspect is evaluated, for instance, when the impact of processor's utilization on execution strategies is evaluated, the other parameters are kept as constants.

Utilization Impact

Fig. 3 shows the performance of different execution strategies when the processor's utilization decreases from 1 to 0.3. As depicted in Fig. 3, the LTF always consumes the most energy. Comparing with the IRCS, the GSSR-UNS-IS consumes less energy when $U > 0.8$ and $U < 0.6$. However, in the middle range, i.e. when $0.7 \leq U \leq 0.8$, these two have the similar behaviors.

The reason is that the IRCS simply assumes all the tasks are fault-protected, hence, under high processor's utilization, the slack time is limited, protecting long tasks will inevitably reduce the opportunities to scale down tasks' working frequencies and hence restricts the energy saving performance. Furthermore, the IRCS searches the suitable frequency for each task in a heuristic manner, when the search space is small, the chance to find a good solution is high. However, under lower processor utilization, more frequency levels are adoptable, which results in larger search space, hence, the chance to hit a good solution is reduced and its energy saving performance decreases.

Rather than always protecting the whole task set, our proposed approach judiciously selects the fault-protected task set. In addition, instead of heuristically searching for tasks' suitable working frequencies, we directly calculate the best processing frequency or neighboring frequencies and assign them to the whole task set. The calculation is independent of the search space. Hence, the proposed GSSR-UNS-IS approach outperforms the IRCS both under high ($U > 0.8$) and low ($U < 0.6$) processor utilization. When $0.7 \leq U \leq 0.8$, GSSR degenerates to GSHR, while the search space is not large enough to deviate the heuristic IRCS from finding a good solution, hence, it has the similar energy saving performance as the GSSR-UNS-IS .

Sensitivity to the Variation of Task Worst-case Execution Times

This set of experiments is to investigate how sensitive each execution strategy is to the variation of task worst-case execution times in a given task set. The simulation results are depicted in Fig. 4. From the figure, we can see that the LTF approach still behaves the worst from energy consumption perspective. However, it is insensitive to the change of the variations of task worst-case execution times. On the other hand,

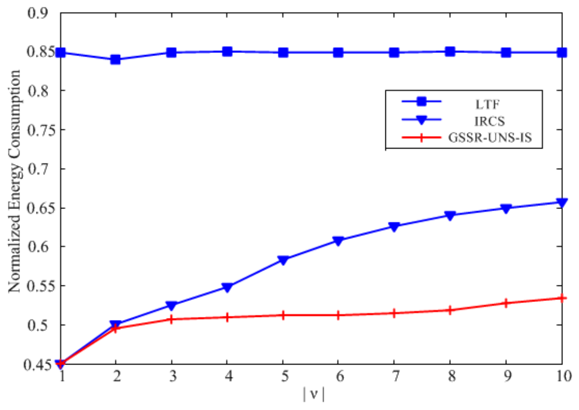

Figure 4: Sensitivity to Variation of Task Worst-case Execution Times ($n = 20, c_{\min} = 20ms, \mathcal{V} = 0.6, R_g = R_0$)

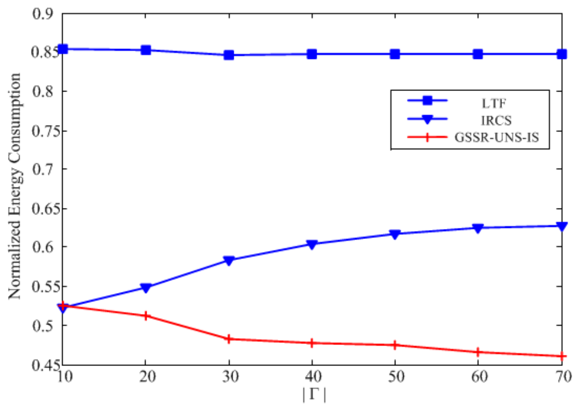

Figure 5: Scalability with Respect to the Size of Task Set ($c_{\min} = 20ms, \mathcal{V} = 4, U = 0.6, R_g = R_0$)

the energy consumption of the IRCS continues growing when the value of $\mathcal{V}$ increases, although the GSSR-UNS-IS based approach also consumes more energy under larger $\mathcal{V}$, but compared to the IRCS, the increase rate is much slower.

When $\mathcal{V} = 1$, the IRCS and our proposed approach perform the same. However, when the variation of task worst-case execution times becomes large, for instance, when $\mathcal{V} = 10$, GSSR-UNS-IS can save as much as 13% more energy than the IRCS.

Scalability with Respective to the Size of Task Set

Fig. 5 shows the comparison by varying the number of tasks. The energy consumption of the LTF approach is not sensitive to task set size, but it always consumes the most energy. However, when the number of tasks increases, even when the total processor utilization remains the same, it impacts the energy saving performance of both IRCS and our GSSR-UNS-IS approaches, but in different directions. More specifically, when the number of tasks increase, IRCS consumes more energy while GSSR-UNS-IS can save more. For instance, when the number of tasks is set to 70, the GSSR-UNS-IS approach can save about 15% more energy when compared to the IRCS.

This is still due to the limitations of the heuristic IRCS as

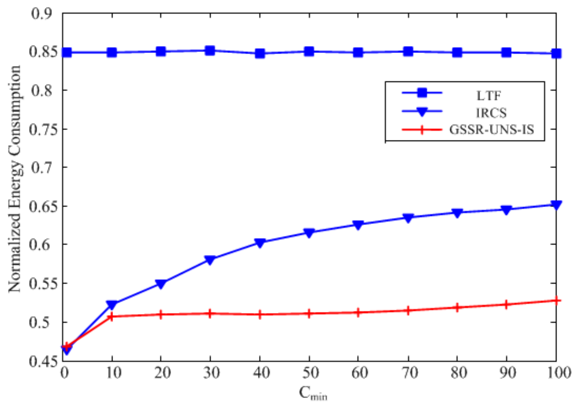

Figure 6: Scalability with Respective to Size of Task ($n = 20, \mathcal{V} = 4, U = 0.6, R_g = R_0$)

addressed above, i.e. increasing the task set size expands the search space, hence results in the performance degradation.

Scalability with Respective to the Size of Task

In this experiment, we fix the variation of task worst-case execution times $\mathcal{V} = 4$, and change the $c_{\min}$ from 1ms to 100ms. Hence, all tasks' WCETs change accordingly. From Fig. 6, we can see that the energy consumption of IRCS increases much faster than GSSR-UNS-IS when $c_{\min}$ increases. When $c_{\min} =$ 1ms, GSSR-UNS-IS and IRCS have the same energy consumption, while when $c_{\min} =$ 100ms, IRCS consumes 13% more energy than GSSR-UNS-IS . LTF still consumes more energy, but it is insensitive to the variation of task execution times.

Sensitivity with Respective to the Reliability Requirement

This set of the experiments is to evaluate the proposed approach's sensitivity to the reliability requirement R_g change. As R_0 is defined as the reliability of the task set when all the tasks are executed under $f_{\max}$, then $1 - R_0$ is the corresponding probability of failure during the execution of task set. We scale the probability of failure to vary the reliability requirement, in particularly, we set $R_g = 1 - (1 - R_0)/S$, where S is scaling factor. As LTF only works when $R_g = R_0$, hence we exclude it from this set of experiments.

The results are shown in Fig. 7. From the figure, we can observe when S increases, i.e. the reliability requirement R_g increases, both the IRCS and GSSR-UNS-IS approaches consume more energy. This is due to the need for reserving more recovery blocks to meet the target reliability, hence less slack time is available for frequency scaling to save energy. With respect to the energy consumption increasing rate, both of the approaches have about the same energy consumption increasing rate.

7. Conclusion

Reliability and power/energy conservation are two of the most critical design issues in today's real-time system design. However, they are often at odds in system resource usage. In this paper, we have developed a new resource sharing technique

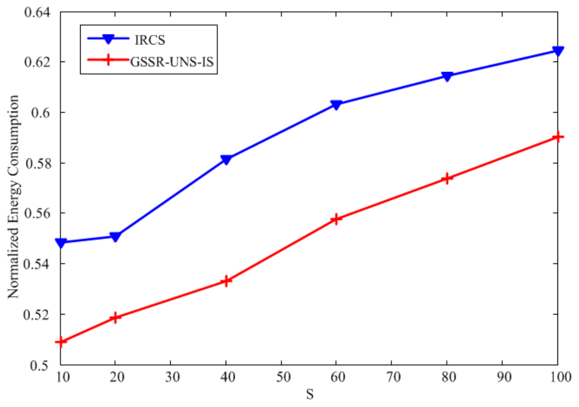

Figure 7: Sensitivity to Reliability Requirement ($n = 20, \mathcal{V} = 4, U = 0.6, R_g = 1 - (1 - R_0)/S$)

and run-time task set execution strategy to guarantee system's reliability and minimize energy consumption. We also have shown that executing a given task set with a constant frequency (or two neighboring frequencies if such a constant frequency is not available) within a interval maximizes task set reliability and minimize its energy consumption. This conclusion applies to both independent and dependent task sets. Comparing to the other fault recovery-based approaches existed in the literature, our simulation results have revealed that our proposed approach can save up to 15% more energy saving and is more resilient to system parameter changes, while guarantee the same level of system reliability, Furthermore, as our approach does not require frequent frequency changes for a task set with independent tasks, it works particularly effective on processors where frequency switching overhead is large and not negligible.

Acknowledgement

This work was supported, in part, by NSF CNS 1018731 and NSF Career 0746643.

References

[1] Aydin, H., Devadas, V., Zhu, D., dec. 2006. System-level energy management for periodic real-time tasks. In: Proceedings of 27th IEEE International Real-Time Systems Symposium. RTSS. pp. 313 –322.

[2] Burd, T. D., Brodersen, R. W., 1995. Energy efficient cmos microprocessor design. In: Proceedings of the 28th Hawaii International Conference on System Sciences. HICSS. pp. 288–297.

[3] Casanova, H., Desprez, F., Suter, F., 2010. Minimizing stretch and makespan of multiple parallel task graphs via malleable allocations. In: Proceedings of the 2010 39th International Conference on Parallel Processing. pp. 71–80.

[4] Castillo, X., McConnel, S. R., Siewiorek, D. P., Jul. 1982. Derivation and calibration of a transient error reliability model. IEEE Trans. Comput. 31 (7), 658–671.

[5] Chandra, V., Aitken, R., 2008. Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos. In: Proceedings of the IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems. DFT. pp. 114–122.

[6] Ernst, D., Das, S., Lee, S., Blaauw, D., Austin, T., Mudge, T., Kim, N. S., Flautner, K., 2004. Razor: circuit-level correction of timing errors for low-power operation. Micro, IEEE 24 (6), 10–20.

[7] Hazucha, P., Svensson, C., dec 2000. Impact of cmos technology scaling on the atmospheric neutron soft error rate. Nuclear Science, IEEE Transactions on 47 (6), 2586–2594.

[8] Huang, W., Stant, M. R., Sankaranarayanan, K., Ribando, R. J., Skadron, K., 2008. Many-core design from a thermal perspective. In: Proceedings of the 45th annual conference on Design automation. DAC. pp. 746–749.

[9] Iyer, R. K., Rossetti, D. J., Hsueh, M. C., Aug. 1986. Measurement and modeling of computer reliability as affected by system activity. ACM Trans. Comput. Syst. 4 (3), 214–237.

[10] Jaberi, N., 2012. An open question about dependency of life time of hardware components and dynamic voltage scaling. CoRR abs/1203.3909.

[11] Kim, W., Gupta, M. S., yeon Wei, G., Brooks, D., 2008. System level analysis of fast, per-core dvfs using on-chip switching regulators. In: Processding of International Symposium on High-Performance Computer Architecture. pp. 123 – 134.

[12] Pradhan, D. K. (Ed.), 1986. Fault-tolerant computing: theory and techniques. Vol. 1. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[13] Rizvandi, N. B., Zomaya, A. Y., Lee, Y. C., Boloori, A. J., Taheri, J., 2012. Multiple frequency selection in dvfs-enabled processors to minimize energy consumption. CoRR.

[14] Shafik, R. A., Al-Hashimi, B. M., Chakrabarty, K., 2010. Soft error-aware design optimization of low power and time-constrained embedded systems. In: Proceedings of the Conference on Design, Automation and Test in Europe. DATE. pp. 1462–1467.

[15] Srinivasan, J., S.V., A., P., B., J., R., Hu, C.-K., 2003. Ramp: A model for reliability aware microprocessor design. IBM Research Report, RC23048.

[16] Yang, C.-Y., Chen, J.-J., Kuo, T.-W., Thiele, L., april 2009. An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In: Design, Automation Test in Europe Conference Exhibition. pp. 694 –699.

[17] Zhao, B., Aydin, H., Zhu, D., 2009. Enhanced reliability-aware power management through shared recovery technique. In: Proceedings of the International Conference on Computer-Aided Design. ICCAD. pp. 63–70.

[18] Zhao, B., Aydin, H., Zhu, D., 2011. Generalized reliability-oriented energy management for real-time embedded applications. In: Proceedings of the Design Automation Conference. DAC. pp. 381–386.

[19] Zhu, D., 2006. Reliability-aware dynamic energy management in dependable embedded real-time systems. In: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium. RTAS. pp. 397–407.

[20] Zhu, D., Aydin, H., 2006. Energy management for real-time embedded systems with reliability requirements. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design. ICCAD. pp. 528–534.

[21] Zhu, D., Aydin, H., oct. 2009. Reliability-aware energy management for periodic real-time tasks. Computers, IEEE Transactions on 58 (10), 1382 –1397.

[22] Zhu, D., Aydin, H., Chen, J.-J., 2008. Optimistic reliability aware energy management for real-time tasks with probabilistic execution times. In: Proceedings of the Real-Time Systems Symposium. RTSS. pp. 313–322.

[23] Zhu, D., Melhem, R., Mosse, D., 2004. The effects of energy management on reliability in real-time embedded systems. In: Proceedings of the IEEE/ACM International conference on Computer-aided design. ICCAD. pp. 35–40.